Business Role-Object Specification: A Language for Behavior-aware Structural Modeling of Business Objects

Hendrik Schön[1], Susanne Strahringer[1], Frank J. Furrer[2], and Thomas Kühn[2]

[1] TU Dresden, Business Informatics, esp. IS in Trade and Industry, Dresden, Germany
{hendrik.schoen,susanne.strahringer}@tu-dresden.de
[2] TU Dresden, Software Technology Group, Dresden, Germany
frank.j.furrer@bluewin.ch, thomas.kuehn3@tu-dresden.de

Abstract. Representing and reusing the business objects of a domain model for various use cases can be difficult. Especially, if the domain model is acting as a template or a guideline, it is necessary to map the enterprise's individual structure and processes on the shared domain model. Structural modeling languages often do not meet this requirement of reusing structures and complying to established processes. We propose a modeling language called BROS (Business Role-Object Specification) for describing the business objects' structure and behavior for structural models, based on a given domain model and process models. It utilizes roles for a use case related specification of business objects as well as events as interfaces for the business processes affecting these roles. Thus, we are able to represent and adapt the business object in different contexts with individual requirements, without changing the underlying domain model. We demonstrate our approach by modeling a simple case.

Keywords: Structural Model, Role, Behavior, Business Logic

1 Introduction

Software modeling enables the design, construction, configuration, and checking of static and dynamic parts of a software system, that is the structure and behavior, before implementation. Especially for enterprise information systems, we are in need of modeling new or changed model parts, as enterprises frequently use individual, enterprise-specific business processes. Therefore, during modeling of information systems, one often encounters conflicting goals and has to choose between either following a unified and possibly standardized domain model or adhering to the individual and unique business logic of the targeted system context. On the one hand, software engineers want to stick to the principle of standardization and rely on the unified domain model. On the other hand, stakeholders want their individual requirements to be implemented.

When modeling a software system, the model should have four characteristics: clarity, commitment, communication, and control [1]. Especially the last two aspects are important: "The model truly and sufficiently represents the key properties of the real world to be mapped into IT solutions", and "The model is used for the assessment

14th International Conference on Wirtschaftsinformatik,
February 24-27, 2019, Siegen, Germany

of specifications, design, implementation, reviews and evolution." [1] Both aspects need to be fulfilled if the model is to be effectively used in software construction. However, current models for such software construction are strong in modeling either the static part (like UML class diagrams [2]), or the dynamic part of a system (like BPMN models [3]). To address the "communication" aspect of the model, both parts need to be taken into account to represent the properties of the real world and their mapping into the IT world. However, the new or changed model should avoid divergences from an available or given domain model that is supposed to serve as a basis for many application models. This is important for the "control" property of a model, to be able to assess the specification, design, and implementation of the future software system. Therefore, compliance with the domain model and realization of specific business logic needs to be harmonized.

In this paper, we propose a new modeling language as part of a method to solve this issue. Our modeling language named BROS (Business Role-Object Specification) can be used to describe and extend a software system that is based on an underlying domain model, and explicitly includes process-driven business logic with respect to its effects on the structural model. In general, BROS is a structural modeling language for design time specification of business objects concerning a (maybe given) domain model as well as specific business logic. Roles fulfilling objects cover the static specification part regarding the separation of concerns, whereas the dynamic specification part of the business logic is expressed via events. The final BROS model serves as a blueprint for development and can be implemented in role-based modeling languages. Although the structural part dominates, we strive for a behavior-aware modeling approach.

The remainder of this paper is structured as follows. Section 2 discusses the related work and current approaches of behavior-aware structural modeling. Section 3 gives an overview of the BROS method, whereas in section 4 we describe the BROS language. Section 5 demonstrates the language with a use case of ordering a pizza, followed by the summary and conclusion in section 6.

2 Related Work

As mentioned before, we strive for a behavior-aware structural modeling method with roles and events. We want to address the issues of using single domain models for various use cases as well as model towards the individual business logic of an enterprise. Especially the tailoring of conceptual business objects of the domain model for different process models is an important feature of modeling software systems.

The role-based paradigm has increased its visibility in the last decades [4, 5]. The static "class" as the main object is not sufficient for modeling business concepts because of its limitation on fixated identity, type, and context. Therefore, various authors propose the role as an appropriate modeling construct in business domains [6–8]. Thus, role-based enterprise modeling is not unknown in modeling.

There are various approaches that model enterprise-specific use cases with roles. In the field of ontologies, many authors state the conceptual foundation of role-based modeling (e.g., [9, 10]). However, conceptual foundations lack implementation details

and are only a preliminary stage. In contrast, several works contribute to the methodological aspect of role-based enterprise modeling (e.g., [7, 11]). Kühn et al. present a role meta-model approach of modeling roles for different contexts. However, they do not elaborate in detail on methods of using roles for given business logic.

Colman [12] defines two different ways of representing a role: "player-centric" and "organization-centric", where the latter is focused on the organizational boundary of business logic. Regarding the modeling of information systems, the understanding of "processual roles" is proposed (e.g., in [13]), determining that a role of an object is used as a participant of a certain process. In general, the inclusion of temporal information of business processes is handled in many different ways. Thus, modeling behavior in structural models receives attention in software and system construction. One may use model transformation to solve this issue by transforming process models into structure. However, such approaches do not relate to an underlying domain model as the target structure in general, which is not preferable for template-based system design. A reversed approach includes temporal information into structural models where the modeling of elements like events, situations, and states are in focus. Various papers examine the more abstract, semantic, and conceptual views on events and scenes (e.g., [14, 15]), to increase the foundational understanding of events. However, the details of the method are missing, how one may model an event within a structural model, although that was not the author's intention. Edelweiss et al. [16] propose a method that introduces temporal aspects and object changes to object-oriented models. However, the structural modeling method remains unconsidered. As a relevant related work for BROS, Olivé and Raventós [17] present an approach for defining the concept of an event entity within structural object-oriented models. The authors define the event as a UML stereotype and utilize OCL to formalize post-conditions for events. Nevertheless, by including events into the core model, the reuse of an underlying domain model, as well as modeling towards different business processes, would be difficult. In addition, events as UML stereotypes are still classes, which can be difficult when modeling towards business process models such as BPMN.

3 The BROS Method

Our method was designed to construct and extend software systems by (re)use of domain models. The initial domain model with its business objects may already have been discovered by a domain analysis or given by a domain standard (like in-house templates or reference models). Also, the enterprise-specific business logic should be known. As motivated in the introduction, the BROS method has, in fact, two major core concepts: *(a)* the (re)use of a single domain model to fit different enterprise's business logic, and *(b)* to specify the behavior of objects with respect to the specific business logic. As a result, the final BROS model serves as a blueprint for the construction of an enterprise's software system according to the unique business logic.

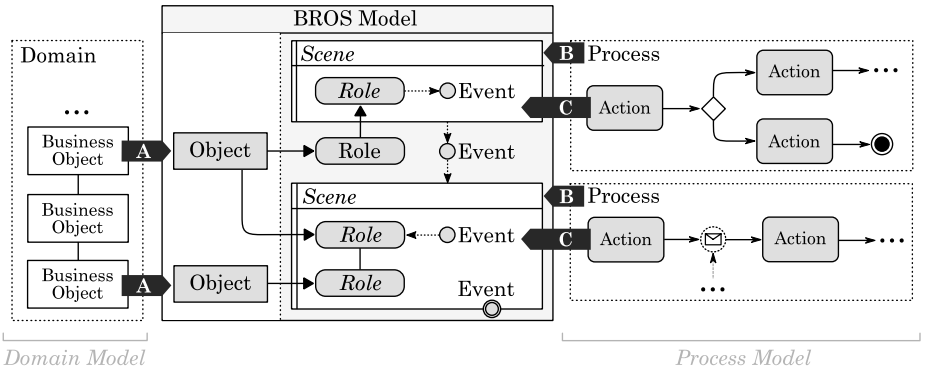


Figure 1. The composition of a BROS model by use of structural and behavioral models

Figure 1 illustrates the BROS method and exemplifies the relationships between the modeling constructs. The objects carry the identity of business objects within the domain ("A" arrows). The (for the enterprise's business logic needed and chosen) objects can be determined by the modeler. Often, the initial domain model is standardized or used by other modelers as well, in which case the modeler must adhere to the domain model as a template, which limits the possibility of introducing new or modifying existent business objects. However, new and changing business logic has to be implemented during the construction and lifetime of the software system. Thus, to express the purpose of (rather static) objects for different (enterprise-specific) business logic in single applications and to support future extensibility, we utilize *roles* as the adaptation construct at design time. A role represents the object and specifies a particular purpose for it. As the objects may be limited to the given domain model, the modeler is free to design roles to fit the objects into the enterprise-specific business logic (e.g., "Meeting Room" as a role of a "Room" object). Even without a limitation to a domain model, roles are useful as a specialization mechanism, since they do not introduce new types and identities for every use case related to the object (e.g., the roles "Meeting Room" and "Dining Room" may use the identity and type of the object "Room" for different use cases). Further, future extensions and evolutions of new functionality are possible by introducing new roles for existent objects.

So far, the specification of objects and roles are the minimum requirements for being a valid BROS model, specifying the static business logic, but not the dynamic behavior. However, for more expressive modeling regarding the enterprise-specific business logic, one can use *scenes* and *events* to describe the behavior and dynamics of roles. A scene is a temporal context for business logic and can often be derived from available process models ("B" arrows). Roles within a scene are acting as participating entities in that process. For example, a "Person in Charge" could be a role within the scene "Room Booking". To specify the lifetime and the participation of roles, we use events as a temporal construct. Events describe a point in time when something has happened (which in turn is based on the enterprise's process model), e.g., "Room Booked". Events are selected by the modeler in such a way as to model the effects of the process on the roles ("C" arrows). The process model that is responsible for the occurrence and

impact of events is called the *background process*. In contrast to the domain model, the final BROS model always uses the background process only as guidance, not as a template. That is also why the processes do not need to be BPMN models but may also be of any other behavior model type, e.g., petri nets or sequence diagrams. An extraction of suitable events out of these model types may, however, be different. In conclusion, the BROS model fills a gap between a generic domain model and the specific business logic by providing a possibility to model towards specific requirements of an enterprise's information system.

4 BROS Modeling Language

In this section, we describe the BROS modeling elements in detail. For a formal representation, the complete BROS modeling language is shown in figure 2. We based our meta-model on [18] and modified it towards our requirements.

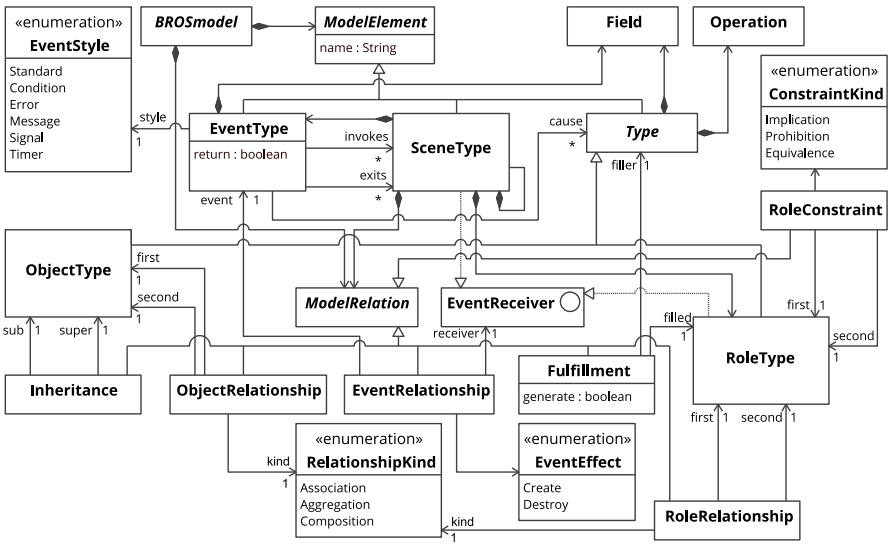


Figure 2. The BROS language meta-model

4.1 Objects

The BROS objects represent the business objects of the model, such as a customer, a rocket, an order, a receipt, or an appointment. The objects are the basic concept for the structural BROS model and are specified using roles, events, and scenes. In theory, we do not need any domain model to start a BROS model. However, if we have such a domain model, we can use its (possibly standardized) business objects as a template for the BROS objects. We define an object as follows:

> *The object is the central modeling construct for representing business objects within a given business domain. The object has an own identity and is uniquely*

identifiable among all other objects. It contains all necessary information to represent the business object with its universally valid properties, behavior, and relationships within the system.

BROS objects are shaped and act similar to the classes from UML. They can be associated with or inherited from other objects and may contain attributes and operations. The reason why we do not call them class but object is simply due to stress that objects always represent a real business object and not any technical class concept. Should the domain model already be in an implementable format (e.g., UML classes with attributes and methods), then we are able to use its classes as BROS objects. However, if this is not the case, one has to create the necessary objects and their relationships in an implementable format.

Please note that the *object* only concerns the object itself. A *compound object* [19], instead, comprises the object as well as all its roles and related events. With the model element of objects, we provide a suitable base for starting the construction of the software system, regardless of whether the object is newly created or already available via an implementable domain model.

4.2 Roles

The definition and understanding of a role in a conceptual model are already well elaborated, e.g., in [20]. In BROS, roles are the static adaptation and configuration parts of business objects. The context may be dynamic (within a scene) or static (the basic BROS model). For both, we use the context definition from Dey [21]:

"Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." [21]

We interpret this definition in such a way, that a scene (and the model itself) is a particular "information" (the "context") that characterizes the situation of objects (the "entities") whereas the objects then are relevant for the general interaction within the scene, expressed as roles as context participants. Our definition of a BROS role is based on the context definition:

A role is a contextual modeling construct with state and behavior that is fulfilled by an object or its roles to represent it in the user's context and extend or change its corresponding specifications and interactions.

A role can be fulfilled either directly by the object or indirectly by a role that is already fulfilled by the object. In our definition, at the type level, the role is "fulfilled by" an object as a *player*. However, at the instance level, the object instance "plays" the role instance (respectively the role is "played by" its player). A fulfilled role shares the related player's identity at runtime. Graphically, a role is represented by a rectangle with curved edges. Figure 3 shows a role fulfillment.

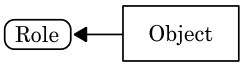


Figure 3. A player object fulfills a role

Role Relationships. In order to successfully represent business logic and unique structure, different relationship kinds are needed to represent the interaction between two elements. Roles can be related to each other to represent their related business logic and interactions, based on the UML language. This includes the kinds shown in table 1.

Further, BROS also supports the usage of role constraints, introduced by [5]. We may use the constraints *Implication*, *Prohibition*, and *Equivalence* to specify constraints between two roles.

Table 1. Role relationship kinds

Name	*Icon*	*Description*
Association	———	interaction-based connection
Composition	◇———	"contains" connection
Aggregation	◆———	"owns" connection (special case of composition)
Implication	·····➤	role A implies playing role B
Prohibition	·····▷	role A and B cannot be played at the same time
Equivalence	⊢·····⊣	role A and B can only be played together

Role-based Adaptation. To redefine several aspects of an object in a specific context, roles can be fulfilled to alter its structure or behavior. For that, we use the delta concept, based on the ideas of delta programming [22]. We only use the *add* (+) and *delete* (–) actions, as those two are logically sufficient to represent all other modifications. For convenience, we also allow *modify* (~) as a combination of both. A role, therefore, may use add, delete or modify as *adaptation actions* to alter properties of its player object. The things we may redefine with roles are manifold: attributes, visibility, methods, parameter, types, or other type-atomic elements. However, we never alter the actual object, but only its roles hold the delta information. In fact, with roles, it is possible to represent (and modify) the original object aligned to the specific requirements of the business logic. Nevertheless, it is not mandatory to specify any delta for using roles. A role may be empty in its delta, but can still be used to express certain business logic, e.g., a state of an object. Note that a given delta implies knowledge about the (future) player and its properties to which the delta refers. If new players are added, the delta may require changes. Figure 4 shows an example of an adaptation.

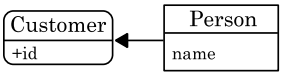


Figure 4. A role changes the player's properties (adds "id" as delta)

4.3 Events

We use events as an essential mechanism for the dynamic aspect of roles. In the UML, events are introduced as "something that may occur at a specific instant in time. One event may have many occurrences, which may happen at different times." [2] We share this definition and extend it further:

> *An event is a modeling construct for specifying the temporal behavior of roles. It occurs at a specific instant of time, with possible reoccurrences. An event may have causes to define the objects that may trigger the event and determines the object fulfillment of roles.*

Event modeling is a frequent topic in conceptual modeling, mainly in process models [23]. However, in BROS, we explicitly focus on the impact of process events on the objects (and their roles) of structural role-based models. Such events are, as described in section 3, interfaces to business process models and represent the impact of process models in the structural model. Events are context-dependent, which means that they only apply in the context and can be triggered by objects or roles in the same scene in which they are located.

When modeling an event, it appears as a simple circle within the diagram. At design time, an event in the model means that it can occur in the course of business logic and certain effects regarding roles result from it. However, at runtime, an event is something that must be triggered, for example, by a method call. As a result of events, roles are created and dropped by the *event effect*. Each role created and orchestrated by an event is called a *dynamic role*, whereas roles that occur independently of events in the system are called *static roles*.

Event Styles. Events are also known from BPMN and described as "something that happens during the course of a process". [3] The BPMN specification differentiates several types of events like message, error, or cancel. As such a classification of events is useful for business modeling, BROS uses some of them within its specification, too: standard, message, timer, error, condition, and signal. Those *event styles* are shortly represented in table 2. However, BROS uses slightly different semantics. Whereas the BPMN event types are focused on the different participant actions, in BROS an event only types the origin of something that happens (e.g., in BPMN, the timer event is involved in an actual workflow, and in BROS it is triggered by an "invisible clock").

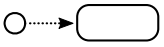
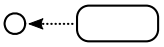
Table 2. General event styles

Name	*Icon*	*Description*	*Example (Pizza Ordering)*
Standard		- standard event	customer added
Message		get message event	order received
Timer		get timer event	pizza 20 minutes baked
Condition		get condition event	pizza less than 40°C

Error		get unexpected event	no more salami available
Signal		get signal event	successful credit card check

Event Effects. An event defines a specific instant in time at which a role instance is created or destroyed. This enables orchestration of the roles at runtime. If an event is triggered, it handles the roles as a consequence (the event's effect). The event's effect options are shown in table 3. Although events can only be triggered within their context, event effects can in principle have roles outside of their own context as a target.

Table 3. Event effects

Name	*Icon*	*Description*
Create		bind a new role to an object
Destroy		unbind a role from an object

4.4 Scenes

The grouping of roles that belong to a specific (temporal) task is called a scene, adapted from [15, 24]. A scene is the main element for describing the execution of any business logic that affects the roles. We define a BROS scene as follows:

> *A scene is an instantiable temporal collaboration context of roles and events, related to the same business logic part.*

Good indications of such scenes are activities or operations, e.g., "student enrollment" or "order pizza". The scene does not model the business process of such tasks but only the role composition and lifetimes. When defining roles for a scene, it is important to always focus on the scene and its required roles as participants, not the existing business objects and the roles they might be able to use. Thus, the background process acts as a hint but does not give a complete answer on the needed roles for scenes. Further, a scene may vary in its granularity. A rather coarse-grained scene would be "book flight" while a fine-grained scene would be "pay with credit card," possibly also as a sub-scene of the former one. However, events are not limited to scenes. Due to readability, the fulfillment arrows do not cross the edge of a scene, but note the target role name, as shown in figure 5. If an object instance is newly created as soon as it takes part in a certain scene, it is marked with a double arrowhead, also shown in figure 5. That arrow is called a *generate fulfillment* and only possible for fulfillments targeting dynamic roles.

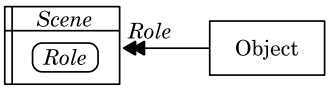
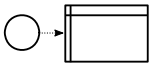
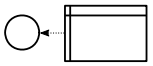


Figure 5. A new object instance is created as soon as the role is created in the scene

Special Event Functions. Scenes also define some other special functions for events and roles to better integrate the business logic flow, which is summarized in table 4.

Table 4. Scene event functions

Event Function	*Description*	*Notation*
Invoker Event	specifies the start of a scene (invokes the scene); all *init roles* of the scene are created	
Exit Event	specifies the termination of a scene from the outside; the scene terminates, and its roles are destroyed; invoked scenes are interrupted (exit)	
Return Event	specifies the end of the scene from the inside; all roles in the scene are destroyed	

A scene may start by any possible *invoker event*. At runtime, multiple instances of a scene, started by possibly different invoker events, may exist. *Init roles*, which occur with the invoker event, are particular roles within a scene that get played (like a parameter) as soon as the scene is invoked. There is always a multiplicity given with a role to determine the number of possible instances for that role at runtime. An init role has a lower bound of at least one instance (e.g., "1..*").

Exit events are events, which are triggered outside of the scene. A triggered exit event is like an external interrupt. All roles within the scene are destroyed, and the scene is over. However, the actual exit event may have some further effects on other roles as well as causes.

The standard way of exiting a scene is using the scene's *return events*, a double lined circle on the scene's edge. They are for the intended outputs of the business logic, even errors or unexpected behavior. The difference between return and exit events is that return events are always triggered within the scene and from the business logic itself, not from the outside. A return event's effect will always take place, regardless of the invoking context. The background process may indicate one or more return events, but it could also be determined by the modeler due to technical reasons.

5 Modeling Case Study

In order to demonstrate our approach, we present a business use case regarding different mechanisms of BROS. As a foundational background process, we use a very simplified "order pizza" BPMN example, as shown in figure 6. It describes a simple pizza ordering within a restaurant plus a routine for customer feedback.

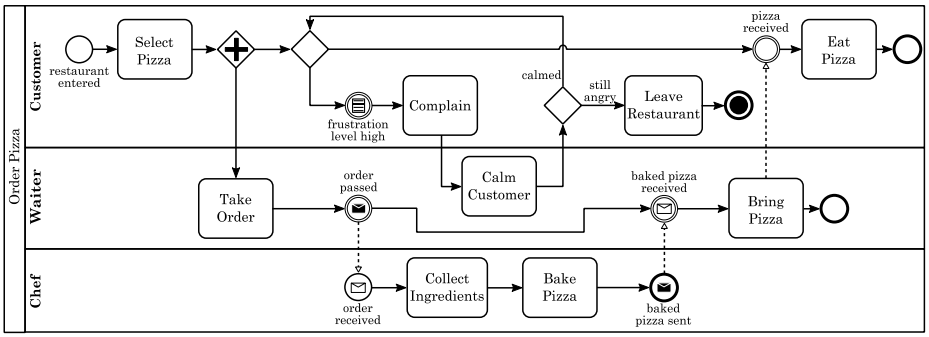


Figure 6. The background process for this case study

It is not possible to translate a BPMN directly into a BROS model. Instead, the construction of BROS models uses BPMN as the background process. We rather assume that a basic structural domain model already exists and is adapted to fit a specific use case or process. Figure 7 shows the final BROS model. We omit attributes and operations due to better readability.

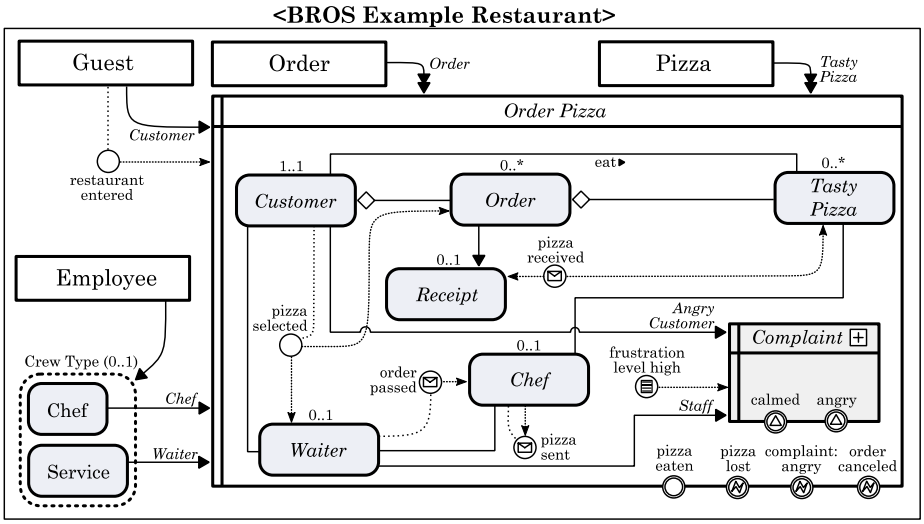


Figure 7. The "Order Pizza" modeled with BROS

Context and Objects. For this example, we use static contexts (the named BROS model), dynamic contexts (scenes), and the corresponding objects out of the business object set given within the domain model. This is either done by the modeler's requirements or by using objects provided by a standardization, e.g., a reference model for the specific domain. We identified the context of the *Order Pizza* model first:

- The background process example describes the pizza ordering process. Thus, our domain is the restaurant, modeled as the *Example Restaurant* BROS model.

- We used four objects: *Employee*, *Guest*, *Order*, and *Pizza*. They serve as the most founded elements and carriers of identities. We assume them to be given within our domain model.
- Our background process implies a scene called *Order Pizza* (probably only one of many scenes within a restaurant). We decided to implement another sub-scene *Complaint* (instead of a single event) due to any existing complexity in the background process related to this.

The additional scene is *folded*, as we are not interested in its inner role behavior but only the players and the outcome. However, other models still might use the full representation of these scenes.

Roles. The roles adapt the objects, tailored for our own (enterprise-specific) business logic of ordering a pizza. Often, the roles can be derived from the given participants of the related background process (e.g., a swim lane in BPMN), which interact with each other in that specific scene. However, one should not forget to also define roles for things and not only for persons.

- We refined the *Employee* object with two static roles: *Service* and *Chef*. Both roles are not part of our scene (means independent of ordering a pizza). We grouped them as *Crew Type*. Every *Employee* has to play at most one role of the *Crew Type*, marked by the multiplicity.
- The *Order Pizza* scene has the following dynamic roles to fulfill its desired business logic: *Customer*, *Waiter*, *Order*, *Chef,* and *Tasty Pizza*. The *Customer* is an init role.
- One needs to define players for the scene's roles. Thus, we use given objects as players. Further, *Order* and *Pizza* use the generate fulfillment so that it is clear that these generate a new instance when taking on the role within the scene.
- Since *Complaint* is folded, we imply knowledge about the inside roles. Thus, a *Waiter* fulfills the role of *Staff* and *Customer* fulfills the *Angry Customer*.
- There is a dynamic role called *Receipt* to respond to the finished order in the form of "pizza received". This role will be played indirectly by the *Order* object.

Please note how the role *Chef* is fulfilling the role of *Chef* again. This is due to the different context. The outside *Chef* has, logically, nothing to do with the *Chef* within the scene. The former role states a crew member, the latter states a person who bakes the pizza. This may be different in other scenes involving the crew's *Chef*. At runtime, both *Chef* roles use the same identity as the related *Employee* instance.

Events. The adding of events depends on the modeler's intention and the background process and can differ widely. We chose one possibility for our use case.

- We stated *restaurant entered* as the invoker event for the *Order Pizza* scene.
- The start and end of a participant on a swim lane in BPMN could provide information about BROS events that start and end a role. For example, we used *order passed* as the start event for the *Chef* role, which ends with *pizza sent*.

- We modeled several return events that end the *Order Pizza* scene. Further, also the returns of the folded scene *Complaint* were defined.
- We used the *Receipt* role as the event effect of *pizza received* to generate the receipt for the customer.
- Finally, optional event causes are added. For example, we modeled that any *pizza sent* event may only be triggered by *Chef*.

Remarks. In general, we used the process, described with BPMN, as a reference for the final structural model, described with BROS. Our BROS implementation is only one possible option. The extent of modeling (and thus the richness of detail and readability) depends on the modeler's requirements and needed scope and granularity. Although the final model may be more complex and larger, it is a detailed adaptation of a (possibly already existing) structural model for a specific business logic use case. Additionally, for example, we could add an "order pizza online" or "payment" scene within the same domain without changing the underlying domain model. Please note that in order to make the model implementable, the (delta) attributes and operations would now have to be added.

6 Conclusion

In the context of role-based modeling, event modeling, and temporal behavior, we have proposed BROS, a new modeling language for behavior-aware structural modeling. It utilizes objects, roles, events, and scenes to describe business objects as role objects, orchestrated by events to represent business logic.

The BROS model reconciles the disparity between a conceptual domain model and individual business logic of the target context. Thus, the BROS method comprises three steps: *(a)* refining of the conceptual model for implementation-ready objects, *(b)* using roles as the variable part of objects for tailoring them towards individual business logic, and *(c)* defining a temporal context regarding business logic in the form of process models, guiding the occurrence of roles and their interaction. We defined a language meta-model and used a fictional case study to show the application of our BROS method in principle and in order to demonstrate feasibility. We consider that the best application of BROS occurs at an early stage of software development when the domain model's expressiveness turns out to be a limitation and software engineers are in need of further details regarding the targeted business logic that has to be fulfilled. On the basis of this, we argue that the possibility of detailed and expressive modeling is helpful for defining new and changing functionality of a software system. Nevertheless, this is also a limitation of BROS, since a BROS model of a comprehensive software system would be too complex. Thus, BROS is possibly better suited for modeling only an excerpt of the complete system, as BROS roles and scenes allow the separation of concerns. However, it is not appropriate to provide a complete system overview. Therefore, a combination of BROS (for detailed views) with a previous model instance (e.g., the overall domain model) works best.

However, some future work is still needed. BROS is currently in its first version. Some modeling scenarios might not yet have been considered, or there might exist some inconsistent features. Further, BROS is formalized by a meta-model, but a concrete syntax (e.g., notation) is also needed. We want to enhance the event element so that more functionality of process models can be covered. The integration of pre- and post-conditions of events also has its benefits [17]. Eventually, our approach needs to be tested in contexts closer to real applications. We also intend to do a broader evaluation and study of the resulting BROS models regarding their applicability and usability (e.g., with [25]) in practice. Additionally, we want to develop tool support and concrete guidelines on how to create BROS models concerning existing background processes.

7 Acknowledgements

This work is funded by the German Research Foundation (DFG) within the Research Training Group „Role-based Software Infrastructures for continuous-context-sensitive Systems" (GRK 1907).

References

1. Furrer, F.J.: Future Proof Software Systems. Software Technology Lecture Talk. Dresden, Germany (2017).
2. Object Management Group: Unified Modeling Language v2.5.1. (2017).
3. Object Management Group: Business Process Model and Notation v2.0. (2011).
4. Steimann, F.: On the representation of roles in object-oriented and conceptual modelling. Data Knowl. Eng. 35, 83–106 (2000).
5. Riehle, D., Gross, T.: Role Model Based Framework Design and Integration. In: Proceedings of the 13th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications, OOPLSA. pp. 117–133. ACM, New York, NY, USA (1998).
6. Cesare, S. de, Henderson-Sellers, B., Partridge, C., Lycett, M.: Improving Model Quality Through Foundational Ontologies: Two Contrasting Approaches to the Representation of Roles. In: Jeusfeld, M.A. and Karlapalem, K. (eds.) Advances in Conceptual Modeling. pp. 304–314. Springer Cham, Stockholm, Sweden (2015).
7. Zhao, L.: Designing Application Domain Models with Roles. In: Assmann, U., Akşit, M., and Rensink, A. (eds.) Model Driven Architecture. pp. 1–16. Springer Berlin Heidelberg, Twente, The Netherlands (2005).
8. Frank, U.: Delegation: An important concept for the appropriate design of object models. J. Object Oriented Program. 13, 13–17 (2000).
9. Almeida, J.P.A., Guizzardi, G., Santos, P.S.J.: Applying and extending a semantic foundation for role-related concepts in enterprise modelling. In: Proceedings of the 12th IEEE International Enterprise Distributed Object Computing Conference, EDOC. pp. 31–40. IEEE (2009).
10. Steimann, F.: Role = Interface : A Merger of Concepts. J. Object-Orientated Program. 23–32 (2001).
11. Zhao, L., Kendall, E.: Role modelling for component design. In: Proceedings of the 33rd Annual Hawaii International Conference on System Sciences. IEEE, Maui, HI, USA (2000).
12. Colman, A.: Role Oriented Adaptive Design, (2006).

13. Boella, G., Steimann, F.: Roles and Relationships in Object-Oriented Programming, Multiagent Systems and Ontologies. In: Cebulla, M. (ed.) Object-Oriented Technology - ECOOP 2007 Workshop Reader. pp. 108–122. Springer Berlin Heidelberg, Berlin, Germany (2007).
14. Galton, A.: States, Processes and Events, and the Ontology of Causal Relations. College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter, UK (2012).
15. Almeida, J.P.A., Costa, P.D., Guizzardi, G.: Towards an Ontology of Scenes and Situations. In: 2018 IEEE Conference on Cognitive and Computational Aspects of Situation Management (CogSIMA). pp. 29–35 (2018).
16. Edelweiss, N., Oliveira, J.P.M. de, Pernici, B.: An object-oriented temporal model. In: Rolland, C., Bodart, F., and Cauvet, C. (eds.) Advanced Information Systems Engineering. pp. 397–415. Springer Berlin Heidelberg, Paris, France (1993).
17. Olivé, A., Raventós, R.: Modeling events as entities in object-oriented conceptual modeling languages. Data & Knowledge Engineering. 58, 243–262 (2006).
18. Kühn, T., Leuthäuser, M., Götz, S.: A Metamodel Family for Role-Based Modeling and Programming Languages. In: Combemale, B., Pearce, D.J., Barais, O., and Vinju, J.J. (eds.) Software Language Engineering. pp. 141–160. Springer, Västerås, Sweden (2014).
19. Leuthäuser, M.: A Pure Embedding of Roles, http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-227624, (2017).
20. Bera, P., Burton-Jones, A., Wand, Y.: Improving the representation of roles in conceptual modeling: theory, method, and evidence. Requirements Eng. (2017).
21. Dey, A.K.: Understanding and Using Context. Pers. Ubiquitous Comput. 5, 4–7 (2001).
22. Damiani, F., Schaefer, I.: Dynamic delta-oriented programming. In: Proceedings of the 15th International Software Product Line Conference, SPLC. ACM, New York, New York, USA (2011).
23. Barros, A., Decker, G., Grosskopf, A.: Complex Events in Business Processes. In: Abramowicz, W. (ed.) Business Information Systems. pp. 29–40. Springer Berlin Heidelberg, Poznan, Poland (2007).
24. Guarino, N., Guizzardi, G.: Relationships and Events: Towards a General Theory of Reification and Truthmaking. In: Adorni, G., Cagnoni, S., Gori, M., and Maratea, M. (eds.) AI*IA 2016 Advances in Artificial Intelligence. pp. 237–249. Springer Cham, Genova, Italy (2016).
25. Krogstie, J.: SEQUAL as a Framework for Understanding and Assessing Quality of Models and Modeling Languages. Int. J. Inf. Syst. Model. Des. 3, 24–45 (2012).